

Efficient Similarity Search in Dynamic Data Streams

Marc Bury, Chris Schwiegelshohn*

Efficient Algorithms and Complexity Theory, TU Dortmund, Germany
`{firstname.lastname}@tu-dortmund.de`

Abstract

The Jaccard index is an important similarity measure for item sets and Boolean data. On large datasets, an exact similarity computation is often infeasible for all item pairs both due to time and space constraints, giving rise to faster approximate methods. The algorithm of choice used to quickly compute the Jaccard index $\frac{|A \cap B|}{|A \cup B|}$ of two item sets A and B is usually a form of min-hashing. Most min-hashing schemes are maintainable in data streams processing only additions, but none are known to work when facing item-wise deletions. In this paper, we investigate scalable approximation algorithms for rational set similarities, a broad class of similarity measures including Jaccard.

Motivated by a result of Chierichetti and Kumar [J. ACM 2015] who showed any rational set similarity S admits a locality sensitive hashing (LSH) scheme if and only if the corresponding distance $1 - S$ is a metric, we can show that there exists a space efficient summary maintaining a $(1 \pm \varepsilon)$ multiplicative approximation to $1 - S$ in dynamic data streams. This in turn also yields a ε additive approximation of the similarity.

The existence of these approximations hints at, but does not directly imply a LSH scheme in dynamic data streams. Our second and main contribution now lies in the design of such a LSH scheme maintainable in dynamic data streams. The scheme is space efficient, easy to implement and to the best of our knowledge the first of its kind able to process deletions.

1 Introduction

Similarity measures between two bit-vectors are a basic building block for many data analysis tasks. Though many algorithms assume a fast, black-box access to similar items to be available, such an assumption is not realistic for large datasets encountered in web-applications. Instead of explicitly computing, storing, and repeatedly querying a similarity matrix, much work has been done on quickly finding the most interesting pairs, which typically are pairs of high similarity. In light of this, a broad range of nearest neighbor search schemes termed locality sensitive hashing have been developed to filter out low similarities from a range of candidate pairs. In this paper, we focus on the Jaccard similarity defined as $\frac{|A \cap B|}{|A \cup B|}$ for two item sets A and B , encountered in a wide range of applications such as clustering [22], plagiarism detection [8], association rule mining [12], collaborative filtering [15] and web compression [11]. Broder [5] introduced min-hashing for fast approximate computation of the Jaccard index between two bit vectors and since then much work has been done on making this approach more scalable and flexible.

*Supported by Deutsche Forschungsgemeinschaft within the Collaborative Research Center SFB 876, project A2

Motivation and Contribution

We study sketching methods for Jaccard based nearest neighbor search in the streaming setting. To be applicable in a big data setting, we will be content to have an approximate solution with stringent storage constraints. More specifically, we assume that we are given n sets of items from a universe consisting of d items and process information on item membership in a streaming fashion. In an insertion only stream, each update consists of a tuple containing a set number j and an item i , stating that i is part of j . At any point in time, we want to be able to report the most similar items. A (semi-)streaming algorithm now aims to process a sequence of updates while using at most $n \text{ polylog}(nd)$ space. The widespread min-hashing approach, a specific example of locality sensitive hashing, can be regarded as such an algorithm, with appealing properties from both a compression and running time perspective. A natural question is whether a similar approach can be developed for more general streaming models such as dynamic streams. Here, updates can also remove an item from a set. Such streams arise naturally when the sets evolve over time. One way to treat such streams theoretically is to study them in the sliding window model, which has been previously done by Datar and Muthukrishnan [17]. In this model, we are interested in only the last W updates of the stream, for some window size W .

In contrast, fully dynamic streams do not remove updates from consideration after a certain progression of time, but require an explicit deletion. In applications where certain item sets change with greater frequency than others, this feature can be more flexible.

In a first step, we show that the Jaccard distance $1 - \frac{|A \cap B|}{|A \cup B|}$ can be $(1 \pm \varepsilon)$ -approximated in dynamic streams. Moreover, the compression used in this approximation is a black-box application of ℓ_0 sketches. This allows for extremely efficient algorithms from both a theoretical and practical point of view. Known lower bounds on space complexity of set intersection prevent us from achieving a compression with multiplicative approximation ratio for Jaccard similarity. This holds even when dropping the computational limitations imposed on streaming algorithms, see for instance [32]. From the multiplicative approximation for Jaccard distance we nevertheless get an ε -additive approximation to Jaccard similarity, which may be sufficient if the interesting similarities are assumed to exceed a given threshold. However, even with this assumption, such a compression falls short of the efficiency we are aiming for, as it is not clear that the relevant similarities can be found more quickly than by evaluating all similarities.

Our main contribution lies now in developing a compression that simultaneously supports locality sensitive hashing while satisfying a weaker form of approximation ratio. The construction is inspired by bit-hashing techniques used both by ℓ_0 sketches and min-hashing. In addition, our approach can be extended to other similarities admitting LSHs other than min-hashing, such as Hamming, Anderberg, and Rogers-Tanimoto similarities. This approach has provable bounds that, despite being weaker than ℓ_0 sketches from an approximation point of view, is extremely simple to implement. Moreover, our experimental evaluation shows the practical compression rates are good, in addition to the fast running times enabled by the standard LSH schemes.

Related Work

Min-Hashing

Min-hashing is the state of the art technique for fast approximate Jaccard similarity search. Roughly speaking, min-hashing computes a fingerprint of a binary vector by permuting the entries and storing the first non-zero entry. For two item sets A and B , the probability that the fingerprint

is identical is equal to the Jaccard similarity of A and B . In practice, a random hash function satisfying certain conditions is sufficient instead of a random permutation of the entries. When looking for item sets similar to some set A , one can arrange multiple fingerprints to filter out sets of small similarity while retaining sets of high similarity, see Cohen et al. [12] and Section 4 for more details.

The approach was pioneered by Broder et al. [5, 8, 6], and has since received much attention in both theory and practice. Many papers focused on the design and analysis of random hash functions, see for instance [7, 24, 19]. While min-wise independent hash functions give the best performance in theory, they are often considered infeasible to store. Thorup [34] showed that the more space efficient 2-wise independent hash functions work well. Other work focused on the efficiency of computing fingerprints. For instance, a faster estimation of similarity is possible by storing the k smallest non-zero entries, see Cohen and Kaplan [13, 14]. Li and König [28] introduced b -bit hashing to further reduce the size of fingerprints.

From a more general perspective, min-hashing is a form of locality sensitive hashing introduced by Indyk and Motwani [25], see also the follow up paper by Gionis, Indyk and Motwani [21] and an overview by Andoni and Indyk [2]. The connection was first drawn by Charikar [9] who gave hashing constructions for related similarity measures. Moreover, he showed that if a similarity measure S admits an LSH, $1 - S$ is a metric. This condition was later shown to be sufficient for rational set similarities by Chierichetti and Kumar [10], see also Theorem 1.

Vector Sketching

Sketching frequency moments and ℓ_p norms of vectors is arguably the most studied problem in theoretical streaming literature. The problem was formally posed in the seminal paper by Alon, Matias and Szegedy [1], which introduced the streaming model and gave upper and lower bounds for a variety of frequency moments and ℓ_p norms of vectors whose entries are continuously modified as the stream progresses. For an even earlier treatment of the related task of approximate counting in a stream we refer to Flajolet and Martin [20]. With respect to space, there exist optimal or nearly optimal algorithms for most values of p in the turnstile model, i. e., in particular for deletions. For the purpose of this paper, the number of non zero elements also known as the Hamming norm¹ (ℓ_0) and the Euclidean norm (ℓ_2) are most relevant. In addition to the space requirements, the best known algorithms for these norms admit constant update times, see Kane, Nelson and Woodruff [27] for ℓ_0 and Thorup and Zhang [35] for ℓ_2 .

The number of distinct elements, a quantity closely related to the Hamming norm, has been previously used by Beyer et al. [4] and Dasu et al. [16] to estimate Jaccard similarity but without being able to process deletions. Recently, Bachrach and Porat [3] reduced Jaccard similarity to the estimation of the second frequency moment (i. e., squared Euclidean norm) if the items are sufficiently similar. This approach can also process deletions with good update times and space bounds, but does not seem to admit a fast locality sensitive hashing scheme, see also Section 2.

¹ ℓ_0 is strictly speaking not a norm but often referred to as such.

2 Outline

Preliminaries

Our item sets are subsets of some universe U of cardinality d . The symmetric difference of two sets $A, B \subseteq U$ is $A \triangle B = (A \setminus B) \cup (B \setminus A)$. The complement is denoted by $\bar{A} = U \setminus A$. A symmetric function $S : U \times U \rightarrow [0, 1]$ with $S(A, A) = 1$ for all $A \in U$ is a similarity. Given $x, y \geq 0$ and $0 \leq z \leq z'$, the rational set similarity $S_{x,y,z,z'}$ between two non-empty item sets A and B is

$$S_{x,y,z,z'}(A, B) = \frac{x \cdot |A \cap B| + y \cdot |\overline{A \cup B}| + z \cdot |A \triangle B|}{x \cdot |A \cap B| + y \cdot |\overline{A \cup B}| + z' \cdot |A \triangle B|}$$

if it is defined and 1 otherwise. We denote nominator and denominator of a rational set similarity by $D(A, B)$ and $N(A, B)$, respectively, i. e., $D(A, B) = x \cdot |A \cap B| + y \cdot (n - |A \cup B|) + z \cdot |A \triangle B|$ and $N(A, B) = x \cdot |A \cap B| + y \cdot (n - |A \cup B|) + z' \cdot |A \triangle B|$. For some arbitrary but fixed order of the elements, we represent A via its characteristic vector $x \in \{0, 1\}^d$ with $x_i = 1$ iff $i \in A$. The ℓ_p -norm of a vector $x \in \mathbb{R}^d$ is defined as $\ell_p(x) = \sqrt[p]{\sum_{i=1}^d |x_i|^p}$. Taking the limit of p to 0, $\ell_0(x)$ is exactly the number of non-zero entries, i. e., $\ell_0(x) = |\{i \mid x_i \neq 0\}|$. An LSH for a similarity measure $S : U \times U \rightarrow [0, 1]$ is a set of hash functions H on U with an associated probability distribution such that for h drawn from H and two item sets $A, B \subseteq U$

$$\Pr[h(A) = h(B)] = S(A, B).$$

We will state our results in a slightly different manner. A (r_1, r_2, p_1, p_2) -sensitive hashing scheme for a similarity measure aims to find a distribution over a family of hash functions H such that for h drawn from H and two item sets $A, B \subseteq U$ we have $\Pr[h(A) = h(B)] \geq p_1$ if $S(A, B) \geq r_1$ and $\Pr[h(A) = h(B)] \leq p_2$ if $S(A, B) \leq r_2$. The former definition due to Charikar [9] is a special case of the latter definition due to Indyk and Motwani [25], though the notions behind both are essentially the same. We choose to phrase our results via the second definition as the lopsided approximation bounds of our algorithms are more easy to present in terms of (r_1, r_2, p_1, p_2) -sensitivity.

Our Approach

The most similar previous work is due to Bachrach and Porat [3]. For any given ℓ_p vector norm, they observed that $\ell_p(x - y)^p = |A \cup B| - |A \cap B|$, where x and y are the characteristic binary vectors of A and B respectively. Provided that $\frac{|A \cap B|}{|A \cup B|} \geq t \geq 1/2$, a sufficiently good estimation of $\ell_p(x - y)^p$ leads to a good estimation of the Jaccard similarity. In principle, any ℓ_p sketch could then be used to estimate the above quantity. By employing the most efficient ℓ_2 sketch available [35], Bachrach and Porat obtained a $(1 \pm \varepsilon)$ -approximation to the similarity of two items with d dimensional features with $O(\frac{(1-t)^2}{\varepsilon^2} \log d)$ bits of space and constant update time when the similarity is assumed to be at least $1/2$.

In this paper, we base our compression around ℓ_0 sketches instead of ℓ_2 . Motivated by the characterization of LSHable rational set similarities of Chierichetti and Kumar [10], we first study approximate estimations of distances $1 - S_{x,y,z,z'}$. It is not too difficult to show that all rational set similarities with metric distances can be $(1 \pm \varepsilon)$ -approximated based on ℓ_0 sketches. Moreover, other ℓ_p sketches do not seem to be able to provide similar guarantees.

In a second step, we aim to provide compressions that can be inputted into an appropriate LSH. The characterization of Chierichetti and Kumar [10] does not imply that the sketched vectors produced by an ℓ_0 approximating algorithm admit an LSH, or even an approximate LSH, nor is this likely to be true. However, all known ℓ_0 sketches retain indexes akin to the fingerprints of min-hashing. These indexes themselves satisfy certain forms of sensitivity. Specifically, we can show that roughly $\log d$ indexes have a lopsided sensitivity guarantee for the scaled Hamming similarity $\frac{|A \cap B|}{d}$ and Rogers-Tanimoto similarity $\frac{|A \cap B|}{d + |A \Delta B|}$, among others.

For other rational set similarities (including Jaccard), the indexes themselves are only sensitive if they have been chosen depending on the cardinalities of two candidate sets A and B . We therefore independently retain indexes for various possible cardinalities for each item set. When we search for item sets similar to some set A , we first filter out all set with too large or too small cardinality and run a LSH on the set of indexes we know to be sensitive. Note that these indexes can be easily identified as we can maintain the exact cardinality of any set in dynamic data stream via counting.

In Section 3 we describe the algorithm and state our bounds formally. We then evaluate our approach experimentally on synthetic and real world data sets in Section 4.

3 Algorithm and Analysis

We start off by showing that any rational set similarity with an LSH can be $(1 \pm \epsilon)$ -approximated in dynamic streams. First, we require the following characterization of such similarity measures.

Theorem 1. *Let $x, y, z, z' > 0$. Then the following three statements are equivalent.*

1. $S_{x,y,z,z'}$ has an LSH.
2. $1 - S_{x,y,z,z'}$ is a metric.
3. $z' \geq \max(x, y, z)$.

(1) \Rightarrow (2) was shown by Charikar [9], (2) \Rightarrow (1) was shown by Chierichetti and Kumar [10] and (2) \Leftrightarrow (3) was proven by Janssens [26]. With this characterization, we are able to prove our first result.

Theorem 2. *Given a constant $0 < \epsilon \leq 0.5$, two item sets $A, B \subseteq U$ and some rational set similarity $S_{x,y,z,z'}$ with metric distance function $1 - S_{x,y,z,z'}$, there exists a dynamic streaming algorithm that maintains a $(1 \pm \epsilon)$ approximation to $1 - S_{x,y,z,z'}(A, B)$ with constant probability. The algorithm uses $O(\frac{1}{\epsilon^2} \log d)$ space and each update and query requires $O(1)$ time.*

Proof. We start with the observation that $|A \Delta B| = \ell_0(x - y)$ and $|A \cup B| = \ell_0(x + y)$, where x and y are the characteristic vectors of A and B , respectively. Since $N(A, B) - D(A, B) = (z' - z) \cdot |A \Delta B|$ is always non-negative due to $z' > z$, we only have to prove that $N(A, B)$ is always a non-negative linear combination of terms that we can approximate via sketches. First, consider the case $x \geq y$. Reformulating $N(A, B)$, we have

$$N(A, B) = y \cdot n + (x - y) \cdot |A \cup B| + (z' - x) \cdot |A \Delta B|.$$

Then both nominator and denominator of $1 - S_{x,y,z,z'}$ can be written as a non-negative linear combination of n , $|A \Delta B|$ and $|A \cup B|$. Given a $(1 \pm \epsilon)$ of these terms, we have an upper bound of $\frac{1+\epsilon}{1-\epsilon} \leq (1 + \epsilon) \cdot (1 + 2\epsilon) \leq (1 + 5\epsilon)$ and a lower bound of $\frac{1-\epsilon}{1+\epsilon} \geq (1 - \epsilon)^2 \geq (1 - 2\epsilon)$ for any $\epsilon \leq 0.5$.

Now consider the case $x < y$. Using a different reformulation

$$\begin{aligned} N(A, B) &= (y - x) \cdot (n - |A \cap B|) + xn \\ &\quad + (z' - y) \cdot |A \triangle B| \\ &= (y - x) \cdot |\overline{A} \cup \overline{B}| + xn + (z' - y) \cdot |A \triangle B|, \end{aligned}$$

we can write the nominator as a non-negative linear combination of $|A \triangle B|$, n and $|\overline{A} \cup \overline{B}|$. Dynamic updates can maintain an approximation of $|\overline{A} \cup \overline{B}|$, leading to upper and lower bounds on the approximation ratio analogous to those from case $x \geq y$.

By plugging in the ℓ_0 sketch of Kane, Nelson, and Woodruff [27] and rescaling ε by a factor of 5, the theorem follows². \square

The probability of success can be further amplified to $1 - \delta$ in the standard way by taking the median estimate of $O(\log(1/\delta))$ independent repetitions of the algorithm. For n item sets, we then get the following corollary.

Corollary 1. *Let S be a rational set similarity with metric distance function $1 - S$. Given a dynamic data stream consisting of updates of the form $(j, i, v) \in [n] \times [d] \times \{-1, +1\}$ meaning that $x_i^{(j)} = x_i^{(j)} + v$ where $x^{(j)} \in \{0, 1\}^d$ with $j = 1, \dots, n$, there is a streaming algorithm that can compute with constant probability for all pairs (j, j')*

- a $(1 \pm \varepsilon)$ multiplicative approximation of $1 - S(x^j, x^{j'})$ and
- an ε -additive approximation of $S(x^j, x^{j'})$.

The algorithm uses $O(n \log n \cdot \varepsilon^{-2} \cdot \log d)$ space and each update and query needs $O(\log n)$ time.

We note that despite the characterization of LSHable rational set similarities of Theorem 1, Corollary 1 does not imply the existence of a locality sensitive hashing scheme or even an approximate locality sensitive hashing scheme on the sketched data matrix.

In the following, we will present a simple dynamic streaming algorithm that possesses such a guarantee, albeit with weaker approximation ratios. While a black box reduction from any ℓ_0 sketch seems unlikely, we note that most ℓ_0 algorithm are based on bit-sampling techniques similar to those found in min-hashing. Our own algorithm is similarly based on sampling a sufficient number of item indexes from each item set. Given a suitably filtered set of candidates, these indexes are then sufficient to infer the similarity. Let $S_k \subseteq U$ be a random set of elements where each element is included with probability 2^{-k} . Further, for any item set A , denote $A_k = A \cap S_k$. At the heart of the algorithm now lies the following technical lemma.

²The exact space bounds of the ℓ_0 sketch by Kane, Nelson and Woodruff depends on the magnitude of the entries of the vector. The stated space bound is sufficient for the purposes in this paper.

Lemma 1. Let $0 < \varepsilon, \delta, r < 1$ be constants. Let A and B be two item sets and let $D(A, B)$ and $N(A, B)$ denote the denominator and the nominator of a metric rational set similarity $S(A, B)$ with parameters x, y, z, z' . Note that in $S(A_k, B_k)$ the value of n is replaced by the size of S_k . Then the following two statements hold.

1. If $S(A, B) \geq r$ and $k \leq \log \left(\frac{(\varepsilon/5)^2 \delta r N(A, B)}{\max\{x + y, z' + y, z + y\}} \right)$ we have

$$(1 - \varepsilon)S(A, B) \leq S(A_k, B_k) \leq (1 + \varepsilon)S(A, B)$$

with probability at least $1 - 2\delta$.

2. The probability that $S(A_k, B_k)$ is by factor of $1/(\delta(1 - \varepsilon/5\sqrt{r}))$ larger than $S(A, B)$ is bounded from above by 2δ .

Proof. Let $D_k = D(A_k, B_k)$, $N_k = N(A_k, B_k)$, and $X_i = 1$ iff $i \in S_k$. If $S(A, B) \geq r$ then $D(A, B) \geq rN(A, B)$. Thus, we have $\mathbf{E}[D_k] = D(A, B)/2^k \geq rN(A, B)/2^k$ and $\mathbf{E}[N_k] = N(A, B)/2^k$. Moreover, we can bound $\mathbf{Var}[D_k]$:

$$\begin{aligned} & \mathbf{Var}[(x - y) \cdot |A_k \cap B_k| + y|S_k| + (z - y) \cdot |A_k \triangle B_k|] \\ &= x^2 \sum_{i \in A \cap B} \mathbf{Var}[X_i] + y^2 \sum_{i \in U \setminus A \cup B} \mathbf{Var}[X_i] \\ & \quad + z^2 \sum_{i \in A \triangle B} \mathbf{Var}[X_i] \\ &\leq ((x^2 - y^2)|A \cap B| + y^2 n + (z^2 - y^2)|A \triangle B|) / 2^k \\ &\leq \max\{x + y, z + y, y\} \cdot \mathbf{E}[D_k] \end{aligned}$$

and

$$\mathbf{Var}[N_k] \leq \max\{x + y, z' + y, y\} \cdot \mathbf{E}[N_k].$$

Using Tchebycheff's inequality we have

$$\begin{aligned} \Pr[|D_k - \mathbf{E}[D_k]| \geq \varepsilon \mathbf{E}[D_k]] &\leq \frac{\max\{x + y, z + y, y\}}{\varepsilon^2 \mathbf{E}[D_k]} \\ &\leq \frac{\max\{x + y, z + y, y\} \cdot 2^k}{\varepsilon^2 r N(A, B)}, \end{aligned}$$

and

$$\begin{aligned} \Pr[|N_k - \mathbf{E}[N_k]| \geq \varepsilon \mathbf{E}[N_k]] &\leq \frac{\max\{x + y, z' + y, y\}}{\varepsilon^2 \mathbf{E}[N_k]} \\ &\leq \frac{\max\{x + y, z' + y, y\} \cdot 2^k}{\varepsilon^2 N(A, B)}. \end{aligned}$$

If $k \leq \log \left(\frac{\varepsilon^2 \delta r N(A, B)}{\max\{x + y, z' + y, z + y, y\}} \right)$ then both $|D_k - \mathbf{E}[D_k]| \leq \varepsilon \mathbf{E}[D_k]$ and $|N_k - \mathbf{E}[N_k]| \leq \varepsilon \mathbf{E}[N_k]$ hold with probability at least $1 - 2\delta$. Then we can bound $S(A_k, B_k) = D_k/N_k$ from above by

$$\frac{D(A, B)/2^k + \varepsilon D(A, B)/2^k}{N(A, B)/2^k - \varepsilon N(A, B)/2^k} = \frac{1 + \varepsilon}{1 - \varepsilon} \cdot S(A, B).$$

Analogously, we can bound $S(A_k, B_k)$ from below by $\frac{1 - \varepsilon}{1 + \varepsilon} \cdot S(A, B)$. Applying a union bound on both events and rescaling ε as in the proof of Theorem 2 concludes the proof of the first statement.

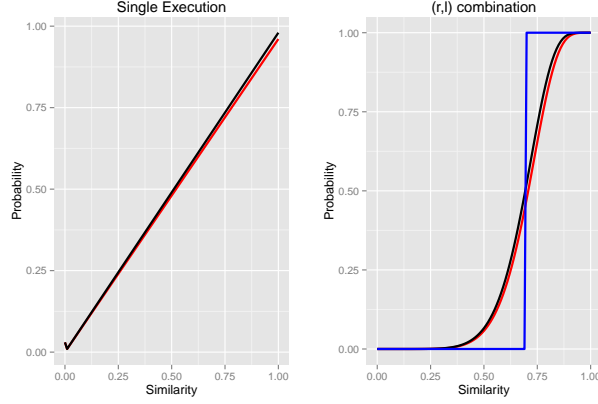


Figure 1: Selection probability (lower and upper bound) vs similarity for parameters $\delta = \varepsilon = r_1 = 0.01$, perfect hash probability of 0.99, $r = 7$, and $l = 10$. The threshold for the step function in the right figure is 0.7.

For the second statement, we can not use Tchebycheff's inequality for bounding the probability that D_k is too large as the expectation of D_k can be very small because we have no lower bound on the similarity. But it is enough to bound the probability that D_k is greater than or equal to $(1/\delta) \cdot \mathbf{E}[D_k]$ by δ using Markov's inequality. With the same arguments as above, we have that the probability of $N_k \leq (1 - \varepsilon') \cdot \mathbf{E}[N_k]$ is bounded by $\frac{(\varepsilon/5)^2 r \delta}{\varepsilon'^2}$ which is equal to δ if $\varepsilon' = \varepsilon/5 \cdot \sqrt{r}$. Putting everything together we have that

$$S(A_k, B_k) \leq \frac{1/\delta}{1 - (\varepsilon/5) \cdot \sqrt{r}} \cdot S(A, B)$$

with probability at least $1 - 2\delta$. \square

For practical purposes, S_k does not have to be a fully random chosen set of items. Instead, we may use a universal hash function. The only parts of the analysis that could be affected are the bounds on the variances, which continue to hold if the hash function is pairwise independent. Applying this lemma on a few better known similarities gives us the following corollary. More examples of rational set similarities can be found in Naish, Lee and Ramamohanarao [31].

Corollary 2. *For the following similarities the given values of k are sufficient to apply Lemma 1:*

$$\begin{aligned} (\text{Jaccard}) \quad & x = 1, y = 0, z = 0, z' = 1 \quad k = \log(\varepsilon^2 \delta r |A|) \\ (\text{Hamming}) \quad & x = 1, y = 1, z = 0, z' = 1 \quad k = \log\left(\frac{\varepsilon^2 \delta r d}{2}\right) \\ (\text{Anderberg}) \quad & x = 1, y = 0, z = 0, z' = 2 \quad k = \log\left(\frac{\varepsilon^2 \delta r |A|}{3}\right) \\ (\text{Rogers-Tanimoto}) \quad & x = 1, y = 1, z = 0, z' = 2 \quad k = \log\left(\frac{\varepsilon^2 \delta r d}{3}\right). \end{aligned}$$

For Hamming and Rogers-Tanimoto similarities, this is already sufficient to run a black box LSH algorithm if the number of sampled items are chosen via Corollary 2. For Jaccard and Anderberg similarities, the sample sizes depend on the cardinality of A , which requires an additional

preprocessing step. The following algorithm and analysis has a particular focus on Jaccard, but can be generalized to work for any metric rational set similarity with $x \geq y$. The case $x < y$ requires further modifications. Since we are not aware of any metric rational set similarities with practical applications in which $x < y$, we omit this description.

For each item, we maintain the cardinality, which can be done exactly in a dynamic stream via counting. If the sizes of two items A and B differ by a factor of at least r_1 , i. e., $|A| \geq r_1 \cdot |B|$, then the distance between these two sets has to be

$$1 - S(A, B) = \frac{|A \triangle B|}{|A \cup B|} \geq \frac{|A| - |B|}{|A|} \geq 1 - 1/r_1.$$

As a first filter, we discard any item set with cardinality not in the range of $[r_1 \cdot |A|, \frac{1}{r_1}|A|]$. Further, for each item set, we retain a sample of indexes for all values of k as described in Lemma 1.

If k is too small, this might result in a large, i. e., infeasible number of indexes to store. However, if k is of order $\log |A|$ or larger, the number of non-zero sampled indexes will not exceed some constant c . Therefore, we use an additional hash function hashing the sampled indexes into $[c^2]$, buckets, where each bucket contains the sum of the entries hashed to it. For the interesting values of k , these entries will then be perfectly hashed and for the smaller values, the space is reduced to a constant. A similar technique was used by Kane, Nelson and Woodruff [27] to maintain a (rough) constant approximation to the ℓ_0 -norm of a vector over the entire stream. For a pseudocode of this approach, see Algorithms 1 and 2.

Algorithm 1: Filter-Preprocessing

input : Parameters $c \in \mathbb{N}$

output: $B_{k,l}^{(j)}$ with $j \in [n], k \in [0, \dots, \log d], l \in [c^2]$

- 1 Initialize $s_j = 0$ for all $j \in [n]$, $B_{k,l}^{(j)} = 0$ for all $j \in [n], k \in [0, \dots, \log d], l \in [c^2]$.
- 2 Let $h : [d] \rightarrow [d]$ be a universal hash function.
- 3 Let $h_k : [d] \rightarrow [c^2]$ be independent universal hash functions with $k = 0, \dots, \log d$.
- 4 Let $S_k = \{i \in [n] \mid \text{lsb}(h(i)) = k\}$ with $k = 0, \dots, \log d$.

On update (j, i, v) :

- 5 $k = \text{lsb}(h(i))$
 - 6 $B_{k,h_k(i)}^{(j)} = B_{k,h_k(i)}^{(j)} + v$
-

Theorem 3. *Let $0 < \varepsilon, \delta, r_1, r_2 < 1$ be parameters. Given a dynamic data stream over n item sets from a universe of cardinality d , there exists an algorithm that maintains a $(r_1, r_2, (1-\varepsilon)r_1, r_2/(\delta(1-\varepsilon/5\sqrt{r_1})))$ -sensitive LSH for Jaccard similarity with probability $1 - \delta$. The algorithm uses $O(n \cdot \varepsilon^{-3} \cdot r_1^{-6} \cdot \delta^{-2} \cdot \log d)$ space.*

Proof. Fix items sets A and B and let j, j' be the corresponding indices for set A and B , respectively. Set $p = (\varepsilon/5)^2 \cdot r_1 \cdot \delta$. If $\text{sim}(A, B) > r_1$ then $r_1 \leq |A|/|B| \leq 1/r_1$. Let S_k be a subset of indices as determined by line 4 of Algorithm 1 and $A_k = S_k \cap A, B_k = S_k \cap B$. For some fixed $k \leq \log(p \cdot |A|)$, denote the event E that $|A_k \cup B_k| \leq 1/(p \cdot r_1) + 1/(p \cdot r_1^2)$ which holds with probability $1 - 2\delta$, see also Lemma 1 and Corollary 2. By setting the number of buckets in the order of $c^2 = \frac{1}{\sqrt{\delta}} (1/(p \cdot r_1) + 1/(p \cdot r_1^2))^2 \in O(1/(\delta \cdot p^2 \cdot r_1^4))$ and conditioning under E , the elements

Algorithm 2: Filter candidates

input : Thresholds $0 < r_1, p < 1$, $B_{k,l}^{(j)}$ from Algorithm 1 with
 $k \in \{0, \log(1/r_1), 2\log(1/r_1), \dots, \log d\}$
output: Set of candidate pairs
1 Let $I = \{0, \log(1/r_1), 2\log(1/r_1), \dots, \log d\}$
2 Let H_j be an empty list for $j \in I$.
3 **foreach** $j \in [n]$ **do**
4 $s = \ell_0(x^{(j)})$
5 **foreach** $k \in [\log(r_1^2 \cdot p \cdot s), \log(p \cdot s)] \cap I$ **do**
6 Add $(j, \text{MinHash}(B_{k,\bullet}^{(j)}))$ to H_k .
7 **return** $\{(j, j') \mid \exists k : (j, h), (j', h') \in H_k \text{ and } h = h'\}$

of $A_k \cup B_k$ will be perfectly hashed by h_k with probability $1 - \delta$ (line 3 of Algorithm 1). Since deleting indices where both vectors are zero does not change the similarity, the similarity of the buckets $B_{k,\bullet}^{(j)}$ and $B_{k,\bullet}^{(j')}$ is equal to the similarity of A_k and B_k . Thus, we have

$$\Pr \left[\text{MinHash}(B_{k,\bullet}^{(j)}) = \text{MinHash}(B_{k,\bullet}^{(j')}) \right] = \text{sim}(A_k, B_k).$$

The theorem then follows by applying Lemma 1 and rescaling δ . \square

Note that if $\text{sim}(A, B) > r_1$ then $\log(|A| \cdot p \cdot r_1) \leq \log(p \cdot |B|)$ and $\log(p/r_1 \cdot |A|) \leq \log(p \cdot |B|)$ which means there are hash values of both sets in some list H_k with $k \in I$ (in Algorithm 2). The parameters in Theorem 3 can be chosen such that we are able to use Algorithm 1 and Algorithm 2 similar to the min-hashing technique in the non-dynamic scenario. This also means that we can use similar tricks to amplify the probability of selecting high similar items in Algorithm 2 and lower the probability in case of a small similarity: Let $r, l \in \mathbb{N}$. Then we repeat the hashing part of Algorithm 2 r times and only add a pair to the output set iff all r hash values are equal. This procedure is repeated l times and the final output set contains all pairs which appear at least once in an output set of the l repetitions. The probability that a pair with similarity s is in the output set is $1 - (1 - p^r)^l$ with $p = (1 - 2\delta)(1 \pm \varepsilon)s$ if $s > c_2$ and $p \leq s/(\delta(1 - \varepsilon/5\sqrt{c_2}))$ otherwise. An example for some fixed parameters is given in Figure 1. Together with Theorem 2 we can approximately compute the distance (or similarity) of the pairs in the candidate set outputted by the described procedure and return a set of pairs with a distance at most T (or similarity of at least $1 - T$) for a known threshold T using $O(n \log n(\varepsilon^{-2} + \log d) + c^2 n \log d)$ space where $c^2 = O(1/(\delta \cdot p^2 \cdot r_1^4))$.

4 Experimental Evaluation

Our primary goal was to measure the quality of the compression computed by Algorithms 1 and 2. We omitted a thorough evaluation of running times, as there exists many papers with experimental evaluations of min-hashing, see Henzinger [23], Cohen et al. [12, 13] and references therein. A particular focus was given on the performance when given little available space. Theorem 2 guarantees us a reasonable approximation to the similarity of each pair, though it is unclear whether this still holds for all pairs simultaneously, especially for small bucket sizes. In addition, we also aimed to find good combinations of bucket size (c^2) and sampling rates (α).

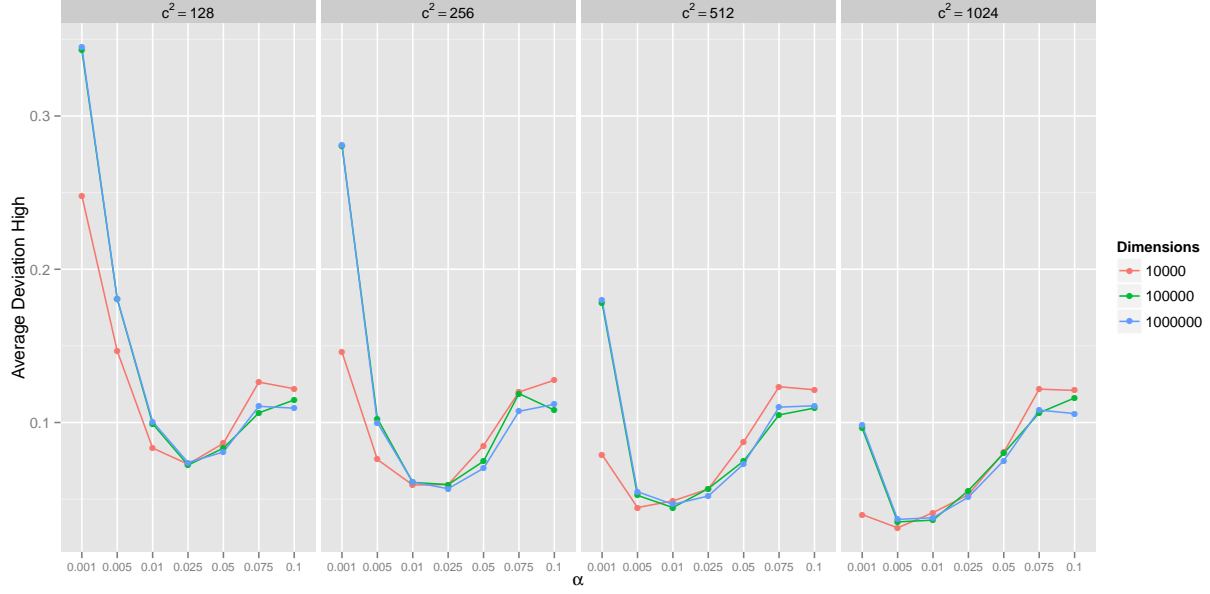


Figure 2: Mean deviation of high-similarity pairs ($S \geq 0.4$) for various parameters of Algorithm 1. Each instance was repeated 10 times.

Setup

We used the following setup for our experiments on both compression and running time. All computations were performed on two identical machines with the same hardware configuration (2.8 Ghz Intel E7400 with 3 MB L2 Cache and 8 GB main memory). The implementation was done in C++ and compiled with gcc 4.8.4 and optimization level 3. Each run was repeated 10 times. Our universal hash functions were generated as in [18] by drawing a non-negative odd integer a and a non-negative integer b . For a given key x , we computed the hash values via $a \cdot x + b$ modulo an appropriate domain. Otherwise the implementation follows that of Algorithms 1 and 2 with various choices of parameters. All random coin tosses were obtained from the random library ³.

Algorithms and Parametrization

Our compression has two choices of parameters, namely the number of buckets c^2 in which we hash (hash function h_k in line 3 of Algorithm 1) and a parameter $\alpha \in (0, 1)$ specifying the relevant values of k in the algorithms. More precisely, for approximating the Jaccard similarity between two items A and B we chose $k = \log(\alpha \cdot (|A| + |B|)/2)$ and output $\text{sim}(A_k, B_k)$. In Algorithm 2, we fixed $r_1 = 1/2$ and added a min-hash value to H_k with $k = s - 1, s, s + 1$ with $s = \lfloor \log(\alpha \cdot |A|) \rfloor$. The analysis indicates that the two parameters cannot be chosen independently of another, but it is nevertheless important to know which range of parameters yield good results. For applications, it is likely that c is chosen to be as large as feasible, and α chosen to yield the best results for a given c . Our ranges for the number of buckets were $c^2 = \{128, 256, 512, 1024\}$ and for the sampling probability $\alpha = \{0.001, 0.005, 0.01, 0.05, 0.075, 0.1\}$.

³<http://www.cplusplus.com/reference/random/>

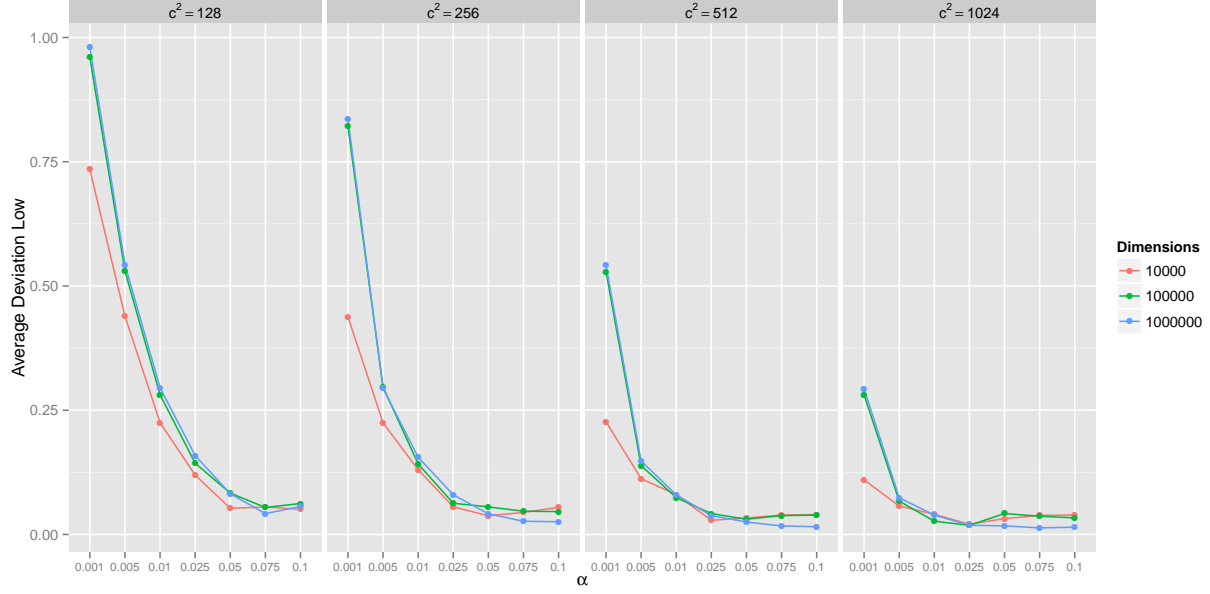


Figure 3: Mean deviation of low-similarity pairs for various parameters of Algorithm 1. Each instance was repeated 10 times.

Further, we implemented a locality sensitive hashing scheme for a faster evaluation of the Jaccard similarity, see also Cohen et al. [12]. We hashed the items and retained for each item set the smallest k indexes associated with items contained in the respective item set. In a second step, we partitioned the k indexes into l groups of r indexes. For each group, we produced a new key by concatenating the indexes and hashing the keys. With an appropriate choice of r and l , we have with good probability that two similar items will have at least one group with identical keys. For our sketches, we used the parameters that yielded the best compression results for the synthetic data sets.

Datasets

An ideal evaluation of our algorithms would use data sets processed via a dynamic data stream, i. e., with insertions and deletion of items. While such streams frequently occur in practice, see for instance Mislove et al. [30], we only had access to final data sets. It could have been conceivable to create a synthetic dynamic stream along with dynamic data. We decided against this for multiple reasons. Firstly, we have little knowledge on the properties of dynamic stream in practice and therefore had no starting point to generate a benchmark. We are also not aware of any existing benchmark. Secondly, our algorithm is, to the best of our knowledge, the only algorithm applicable in this setting and therefore would have no other algorithm to compare to. Moreover, there is no technical difference between processing updates and deletions for our algorithm. If the final data set is identical, the only difference between an insertion only stream and a dynamic stream will be the respective length. As a result, our evaluation only considered final data.

We evaluated our approach both on synthetic data as well as on real-world data from an application using the Jaccard index as a similarity measure. For the synthetic data, we used

Similarity	0.0	0.05	0.1	0.15
Number of pairs	0	0	995	33864
Similarity	0.2	0.25	0.3	0.35
Number of pairs	364496	206572	233303	576286
Similarity	0.4	0.45	0.5	0.55
Number of pairs	861799	593181	549257	144769
Similarity	0.6	0.65	0.7	0.75
Number of pairs	33093	27777	42181	23185
Similarity	0.8	0.85	0.9	0.95
Number of pairs	2617	7287	51042	113886

Table 1: Distribution of similarity values from the G DATA data set.

the benchmark by Cohen et al. [12]. Here we are given a large binary data-matrix consisting of 10,000 rows and either 10,000, 100,000 or 1,000,000 columns. The rows corresponded to item sets and the columns to items, i.e., we compared the similarities of rows. Since large binary data sets encountered in practical applications are sparse, the number of non-zero entries of each row was between 1% to 5% chosen uniformly at random. Further, for every 100th row, we added an additional row of with higher Jaccard similarity in the range of $\{(0.35, 0.45), (0.45, 0.55), (0.55, 0.65), (0.65, 0.75), (0.75, 0.85), (0.85, 0.95)\}$.

To obtain such a pair, we copied the preceding row (which was again uniformly chosen at random) and uniformly at random flipped an appropriate number of bits, e.g., for 10,000 items, row sparsity of 5%, and similarity range (0.45, 0.55) we deleted an item contained in row i with probability $1/3$ and added a new item with probability $\frac{1}{19} \cdot \frac{1}{3} = \frac{1}{57}$.

The real-world data consists of features extracted from so-called PE files donated to us from G DATA ⁴. The dataset is based on 2781 PE file samples from [33] where they were used for clustering to detect malware. G DATA extracted 714 categorical features which we converted to 18359 binary features. Each row in the final matrix has a support of 100-200 entries each. The distribution of similarities for this data set can be found in Table 1.

Results

For compression we compared the exact similarities of the synthetic data set with the approximate similarities obtained for various parameterizations of our algorithm. We did not use min-hashing or any other filtering scheme to quickly evaluate the similarities on either original data set or our compression, as this introduces additional errors. As a result, the time required to evaluate the similarities is of secondary importance, though we did compare the time required to compute the approximate similarities with the time required to compute the similarities on the original data set.

The major importance for this run of experiments was to find good combinations of c^2 and α and comparing the similarities.

We measured the absolute deviation for high similarity (≥ 0.2) and low similarity (< 0.2) pairs separately. For a fixed bucket size, a larger value of α led to fewer items picked, while a smaller value of α led to many hash collisions. In the former case, the compression performed worse on high-similarity pairs, while in the latter case the compression rates for both high and low-similarity

⁴<https://www.gdatasoftware.com/>

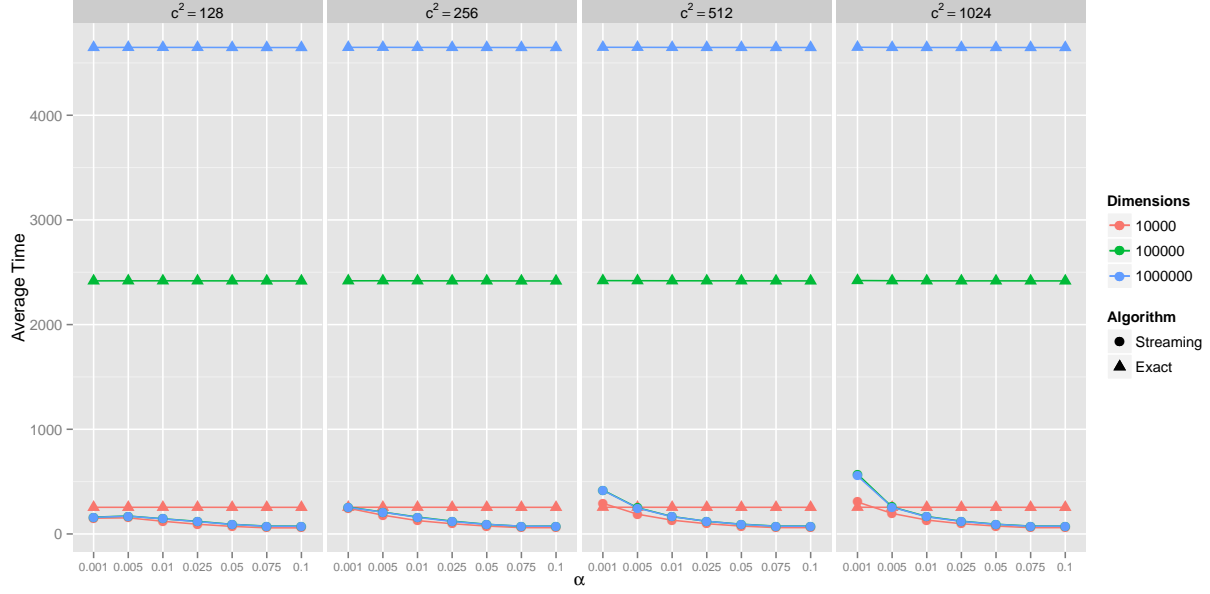


Figure 4: Runtime comparisons of the exact evaluation of the similarities on our summary and on original data set. The summary running times are the mean values of 10 repetitions.

pairs deteriorated, more so for the latter than the former, see also Figures 2 and 3.

The best combinations of c^2 and α were different for high-similarity and low-similarity pairs. Good trade offs between both were achieved for 128 buckets with a sampling rate of $\alpha = 0.05$, 256 buckets with a sampling rate of $\alpha = 0.025$, 512 buckets with a sampling rate of $\alpha = 0.01$, and 1024 buckets with a sampling rate of $\alpha = 0.005$. On these average total deviation for these parameters was always below 0.1 and further decreased for larger bucket sizes. We note that these values of c^2 are below the theoretical bounds of Theorem 3, while having little to acceptable deviation for appropriately chosen values of α .

The time required to compute the sketch and thereafter evaluate the similarities was usually faster by a large magnitude (up to a factor of 10) compared to the original data set, see Figure 4. Generally, the fewer buckets and the lower the sampling rate, the faster the computation was carried out on the sketch. It should be noted that this already holds for relatively sparse data and since the sketch size is independent of the density, we would expect the improvement in time to be more apparent for denser data.

Lastly, we ran the LSH for the G DATA data set, again for various choices of parameters. Unlike for the synthetic data, there is no obvious correct threshold above which the relevant similarities lie. As a general rule, a large value of r moved the threshold towards 1, while a larger value of l moved the threshold to 0. By increasing both, the slope of the similarity curve increases. For a target threshold, i. e., fixed r and l and a fixed sampling rate α , the approximation to the theoretical similarity curve improved with an increasing number of buckets, see Figure 5. For a fixed bucket size, we made a similar observation when varying the sampling rates, see Figure 6.

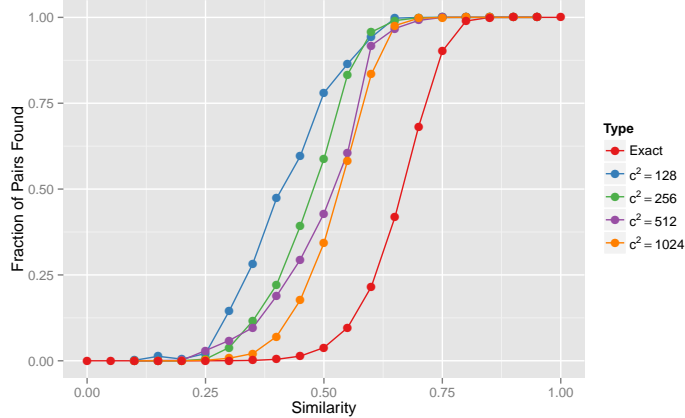


Figure 5: Thresholding for similarities at $r = 10$, $l = 40$ and $\alpha = 0.025$. With increasing bucket size, the theoretical curve marked in red was closer approximated by LSH computed on the sketch.

5 Conclusion and Future Work

In this paper, we have described multiple space efficient data structures for a variety of approximate similarity measures, including Jaccard. Crucially, the data structures can be maintained over data streams with deletions, which has (thus far) rarely been found in literature.

In addition to the good compression and approximation bounds from a more theory driven approach, we also developed a compression algorithm whose output can be processed by any locality sensitive hashing scheme. The compression itself can be computed very quickly, and it scales well to any space restriction. In particular, our experiments show that even with very small space requirements, the mean deviation does not suddenly deteriorate, but slowly increases. Due to its simplicity and its performance in our experimental evaluation, we believe that this algorithm can be used in practical applications.

For future work, it would be especially interesting to develop a realistic model for generating dynamic streams. Not only would such a benchmark make an experimental evaluation of dynamic streaming algorithms easier, it might also result in new frameworks that can be analyzed theoretically. For instance, if items are deleted adversarially, the algorithm will usually have to resort to some form of linear sketch [29]. If items are deleted randomly from the available set of items, other algorithms might be conceivable.

References

- [1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
- [2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.
- [3] Y. Bachrach and E. Porat. Fingerprints for highly similar streams. *Inf. Comput.*, 244:113–121, 2015.

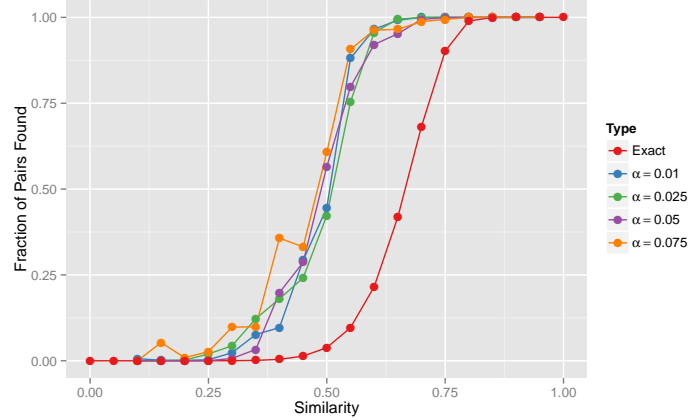


Figure 6: Thresholding for similarities at $r = 10$, $l = 40$ and $c^2 = 256$. The approximation tends to worsen with increasing α .

- [4] K. S. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *Proc. ACM SIGMOD*, pages 199–210, 2007.
- [5] A. Z. Broder. On the resemblance and containment of documents. In *Proc. SEQUENCES, SEQUENCES '97*, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society.
- [6] A. Z. Broder. Identifying and filtering near-duplicate documents. In *Proc. CPM*, pages 1–10, 2000.
- [7] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.
- [8] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.
- [9] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. ACM STOC*, pages 380–388, 2002.
- [10] F. Chierichetti and R. Kumar. LSH-preserving functions and their applications. *J. ACM*, 62(5):33, 2015.
- [11] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *Proc. ACM KDD*, pages 219–228, 2009.
- [12] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE Trans. Knowl. Data Eng.*, 13(1):64–78, 2001.
- [13] E. Cohen and H. Kaplan. Bottom-k sketches: better and more efficient estimation of aggregates. In *Proc. ACM SIGMETRICS*, pages 353–354, 2007.
- [14] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *Proc. PODC*, pages 225–234, 2007.

- [15] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proc. WWW*, pages 271–280, 2007.
- [16] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proc. ACM SIGMOD*, pages 240–251, 2002.
- [17] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Proc. ESA*, pages 323–334, 2002.
- [18] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19 – 51, 1997.
- [19] G. Feigenblat, E. Porat, and A. Shiftan. Exponential time improvement for min-wise based algorithms. In *Proc. ACM-SIAM SODA*, pages 57–66, 2011.
- [20] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.
- [21] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. VLDB*, pages 518–529, 1999.
- [22] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Inf. Syst.*, 25(5):345–366, 2000.
- [23] M. R. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proc. ACM SIGIR*, pages 284–291, 2006.
- [24] P. Indyk. A small approximately min-wise independent family of hash functions. *J. Algorithms*, 38(1):84–90, 2001.
- [25] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. ACM STOC*, pages 604–613, 1998.
- [26] S. Janssens. *Bell inequalities in cardinality-based similarity measurement*. PhD thesis, Ghent University, 2006.
- [27] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proc. ACM PODS*, pages 41–52, 2010.
- [28] P. Li and A. Christian König. Theory and applications of b -bit minwise hashing. *Commun. ACM*, 54(8):101–109, 2011.
- [29] Y Li, H. L. Nguyen, and D. P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proc. STOC*, pages 174–183, 2014.
- [30] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proc. ACM IMC*, San Diego, CA, October 2007.
- [31] L. Naish, H. J. Lee, and K. Ramamohanarao. A model for spectra-based software diagnosis. *ACM Trans. Softw. Eng. Methodol.*, 20(3):11, 2011.

- [32] R. Pagh, M. Stöckel, and D. P. Woodruff. Is min-wise hashing optimal for summarizing set intersection? In *Proc. ACM PODS*, pages 109–120, 2014.
- [33] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic analysis of malware behavior using machine learning. *J. Comp. Sec.*, 19(4):639–668, 2011.
- [34] M. Thorup. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proc. ACM STOC*, pages 371–380, 2013.
- [35] M. Thorup and Y. Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.